An Opportunistic Approach for Logic Program Analysis and Optimisation Using Enhanced Schema-Based Transformations*

Wamberto W. Vasconcelos**, Norbert E. Fuchs
Institut für Informatik
Universität Zürich
Switzerland
vascon@ifi.unizh.ch, fuchs@ifi.unizh.ch

Abstract. We propose an opportunistic approach for performing program analysis and optimisation: opportunities for improving a logic program are systematically attempted, either by examining its procedures in an isolated fashion, or by checking for conjunctions within clauses that can be used as joint specifications. Opportunities are represented as enhanced schema-based transformations, generic descriptions of inefficient programming constructs and of how these should be altered in order to confer a better computational behaviour on the program. The programming constructs are described in an abstract manner using an enhanced schema language which allows important features to be highlighted and irrelevant details to be disregarded.

1 Introduction

The clean declarative semantics of logic programs hides many performance issues which must be accounted for if the programs are to be executed using the currently available technology. These efficiency issues can be dealt with at an early stage, during the preparation of the program: standard logic programming constructs, also named programming techniques, which guarantee a good computational behaviour to those logic programs incorporating them, can be used. Prolog programming techniques have been extensively studied [Bowles et al, 94; O’Keefe, 90; Sterling & Shapiro, 94]; it has been advocated [Kirschenbaum et al, 94] that these standard practices should be explicitly taught as part of a discipline of methodical logic programming development; and logic programming environments have been implemented [Bowles et al, 94; Robertson, 91] incorporating them.

* This work was carried out within the European Human Capital and Mobility Scheme (contract No. CHRX-CT 93-00414/BBW 93.0268) while the first author was visiting the Institut für Informatik, Universität Zürich.
** PhD Student, Department of Artificial Intelligence, University of Edinburgh, Scotland, U.K., sponsored by the Brazilian National Research Council (CNPq, grant No. 201340/91-7), on leave from UECE/Ceará, Brasil.
A second approach addresses efficiency issues after the program has been devised, trying to optimise the existing code. This involves the analysis and subsequent transformation of a given, possibly inefficient, logic program into an equivalent version with better computational behaviour. An underlying assumption of this approach is that commonly occurring but computationally inefficient constructs in logic programs can be identified and eliminated, preferably in a fully automated way, thus relieving logic programmers of the extra burden of worrying about performance issues. A number of approaches for semi-automatic program transformation have been proposed [Fuchs & Fromherz, 91; Lakhotia & Sterling, 90; Nielson & Nielson, 90; Proietti & Pettorossi, 90].

A third option combines both previous approaches: individual predicates are devised using programming techniques and their interrelations are, whenever possible, improved by a program transformation system. Additional information concerning the intended use of each predicate can be collected via the program development tools and methods employed, thus making the transformation process easier and more sophisticated. This idea is pursued in [Flener, 95; Flener & Deville, 95], in which each procedure is devised by means of program templates employing divide-and-conquer algorithms. In [Vargas-Vera, 95; Vargas-Vera et al., 93] a similar idea is proposed, procedures being developed using Prolog programming techniques. In both approaches more sophisticated forms of program combination can be achieved by employing auxiliary information concerning the intended use of each procedure.

The work presented here fits into the second line of research above: we propose an approach for the systematic analysis of a program, detecting opportunities to improve its computational efficiency. These opportunities are generic descriptions of commonly occurring inefficient programming constructs and of how these should be changed in order to confer a better computational behaviour on the program. Changes to program constructs are depicted by means of transformations relating an inefficient programming construct described in a generic, schematic fashion to its modified, more efficient, version. A given program is systematically scanned and portions of its code are matched against the description of inefficient constructs of the transformations: if the matching occurs then the prescribed new version appropriately replaces the previous portions of code and further opportunities in the program are searched for. Each transformation is formulated so as to describe the programming constructs in an economic yet general fashion. We propose an enhanced version of Gegg-Harrison's schema language [Gegg-Harrison, 91] for this purpose, following the proposal in [Fuchs & Fromherz, 91].

In the next section we explain our enhanced schema language. In Section 3 we describe how enhanced program schemata are used to guide program transformations. In Section 4 we propose a realistic scenario in which transformations can be used to analyse and improve the computational performance of a program. In Section 5 we draw some conclusions and show directions for research.